ELSEVIER

Theoretical Computer Science 266 (2001) 249–272

Theoretical Computer Science

www.elsevier.com/locate/tcs

A typed context calculus

Masatomo Hashimoto*,[1], Atsushi Ohori[2]

Research Institute for Mathematical Sciences, Kyoto University, Sakyo-ku, Kyoto 606-8502, Japan

Received August 1997; revised January 2000; accepted February 2000

Abstract

This paper develops a typed calculus for contexts i.e., lambda terms with "holes". In addition to ordinary lambda terms, the calculus contains *labeled holes*, *hole abstraction* and *context application* for manipulating first-class contexts. The primary operation for contexts is *hole-filling*, which captures free variables. This operation conflicts with substitution of the lambda calculus, and a straightforward mixture of the two results in an inconsistent system. We solve this problem by defining a type system that precisely specifies the variable-capturing nature of contexts and that keeps track of bound variable renaming. These mechanisms enable us to define a reduction system that properly integrates β-reduction and hole-filling. The resulting calculus is Church–Rosser and the type system has the subject reduction property. We believe that the context calculus will serve as a basis for developing a programming language with advanced features that call for manipulation of open terms.

Keywords: Context; Lambda-calculus; Type system; Alpha-renaming

1. Introduction

A *context* in the lambda calculus is a term with a "hole" in it. The operation for contexts is to fill the hole of a context with a term. For the purpose of explanation in this section, we write $C[\cdot]$ for a context containing the hole indicated by $[\cdot]$, and write $C[M]$ for the term obtained from $C[\cdot]$ by filling its hole with M. For example, if $C[\cdot] \equiv (\lambda x.[\cdot] + y)\ 3$, then $C[x+z] \equiv (\lambda x.x + z + y)\ 3$. As seen from this simple

* Corresponding author.

E-mail address: masatomo@is.s.u-tokyo.ac.jp (M. Hashimoto).

[1] Current affiliation: Department of Information Science, University of Tokyo, Bunkyo-ku, Tokyo 113-0033, Japan.

[2] Current affiliation: School of Information Science, Japan Advanced Institute of Science and Technology, Tatsunokuchi, Ishikawa 923-1291, Japan. Atsushi Ohori's work was partly supported by the Japanese Ministry of Education Grant-in-Aid for Scientific Research on Priority Area no. 275: "Advanced databases", and by the Parallel and Distributed Processing Research Consortium, Japan.

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00174-2

example, the feature that distinguishes this operation from substitution of the lambda calculus is that it *captures free variables*. In the above example, x in $x + z$ becomes bound when it is filled in the context $(\lambda x.[\cdot] + y)$ 3.

One motivation behind using contexts in the theory of lambda calculus is to study properties of *open* terms. Since the behavior of an open term depends on bindings of their free variables, in order to analyze its behavior, it is essential to consider possible contexts in which the open term occurs. Study of program analyses based on contexts such as observational equivalence [13, 11] yields important results in analysis of programming languages. In these and most of other usages, context is a meta-level notion and its applicability to programming languages has largely been limited to meta-level manipulation of programs. We believe that if a programming language is extended with first-class contexts, then the extended language will provide various advanced features that call for manipulation of open terms. Let us briefly mention a few of them.

Programming environment: In conventional programming environments, programs must first be compiled into "object modules", and they must then be linked together to form an executable program. Moreover, an executable program must be a closed term. If a programming environment can be extended with the ability to link various software components dynamically, then its flexibility will significantly increase. Since the mechanism of contexts we are advocating offers a way of performing linking at runtime, it would provide a basis for developing such an environment in a theoretically sound way.

Distributed programming: In distributed programming, one often wants to send a piece of code to a remote site and execute it there. As witnessed by recently emerging Internet programming languages such as Java [4], this feature will greatly enhance the expressive power of distributed programming. One naive approach to send a program is to pack all the necessary resources as a closure and send the entire closure to the remote site. An obvious drawback to this approach is inefficiency. Since in most cases, communicating sites share common resources such as standard runtime libraries, a better approach would be to send an open term and to make the necessary binding at the remote site. A typed calculus with first-class contexts would provide a clean and type safe mechanism for manipulating open terms.

First-class modules: A program using a module can naturally be regarded as an open term containing free variables whose values will be supplied by the module. One way of modeling a module exporting a set of functions $F_1, \ldots, F_n$ through identifiers $f_1, \ldots, f_n$ would therefore be regarding it as a context that captures variables $f_1, \ldots, f_n$ and bind them to $F_1, \ldots, F_n$, respectively. Using (or "opening") a module then corresponds to filling the hole of the context with the variables. This approach can provide a new foundation for flexible module systems. In conventional languages with modules such as Modula-2 [18] and Standard ML [12], there is rigid separation between the type system for modules and that of terms, and allowable operations on modules are rather limited. Significant potential advantage of the "modules-as-contexts" approach is that modules can be freely combined with any other constructions available in the language,

i.e., that modules are treated as first-class citizens. Needless to say, an actual module system must account for various features such as type abstraction, type sharing and separate compilation, and the above simple view alone does not immediately provide a proper basis for module systems. We nonetheless believe that, when properly refined with various mechanisms for module systems studied in literature, the above approach will open up a new possibility for flexible module systems. Indeed, a recent work by Wells and Vestergaad [17] shows a connection between their module language and our context calculus.

The general motivation for this study is to develop a programming language with first-class contexts that can represent those features in a clean way.

Despite those and other potentially promising features of contexts, a language with first-class contexts has not been well investigated. Lee and Friedman [10] proposed a calculus where contexts and lambda terms are two disjoint classes of objects: contexts are regarded as "source code" and lambda terms as "compiled code". This separation is done by assuming two disjoint variable name spaces: one for lambda terms and one for contexts. As a consequence, in their system, β-reduction and fill-reduction are two disjoint relations without non-trivial interaction. Dami [2] also announced a system for dynamic binding similar to that of Lee and Friedman. While these approaches would be useful for representing source code as a data structure, they do not allow contexts of the language itself to be treated as first-class values inside the language. Kahrs [9] have developed a combinatory term rewriting system that is compatible with contexts. However, contexts and hole-filling themselves are not represented as terms within the system of terms. Talcott [16] developed an algebraic system for manipulating binding structures. Her system includes suitable mechanisms for manipulating contexts. In particular, it contains holes and hole-filling which commutes with substitution. However, this is a meta-level system, and the issue of representing contexts and the associated hole-filling operation inside of the reduction system of lambda calculus is not addressed. One of the features of contexts is to bind variables through holes. In this sense, contexts are closely related to environments. Abadi et al. [1] developed the $\lambda\sigma$-calculus for explicit substitutions. Their motivation is similar in spirit to ours in that it internalizes a meta-level mechanism in the lambda calculus. However, they did not address the problem of first-class treatment of substitutions. In revising the present article, the authors noticed that Sato et al. [14] recently developed an environment calculus where environments are first-class values. In obtaining a confluent calculus, they also address the problem of variable binding in the presence of first-class environments. Their solution to this problem has some similarity to ours, although more general mechanisms are needed for a calculus with first-class contexts. We shall comment on this in some detail when we describe our approach in the next section.

The goal of this paper is to establish a type theoretical basis for a programming language with first-class contexts by developing a *typed context calculus* where lambda terms are simply a special case of contexts. In particular, contexts and lambda terms belong to the same syntactic category sharing the same set of variables, and substitution and hole-filling are defined on the same syntactic objects. This property is essential

for achieving various features explained above. As observed in the literature [9, 10], however, β-reduction and fill-reduction for contexts do not mix well, and a (naive) integration of them yields an inconsistent system. The development of a meaningful calculus containing β-reduction and fill-reduction both acting on the same set of terms constitutes a non-trivial technical challenge. Our main technical contribution is to establish that such a calculus is possible. We prove that the calculus is Church–Rosser and its type system has the subject reduction property.

To obtain a confluent calculus, we have to overcome various delicate problems in dealing with variables, and to introduce several new mechanisms in the lambda calculus. Before giving the technical development, in the next section, we explain the problems and outline our solution.

2. The problem and our solution

It is not hard to extend the syntax of the (untyped) lambda calculus with constructors for contexts. In conventional study, holes in contexts are nameless. However, since our goal is to develop a calculus with first-class contexts, we should be able to consider a context containing other contexts. This requires us to generalize contexts to contain multiple *different* holes, only one of which is filled by each hole-filling operation. One way to define a uniform syntax for those contexts is to introduce *labeled* holes [9]. We use upper case letters $X, Y, \ldots$, for labeled holes. To incorporate operations for contexts as terms in a lambda calculus, we introduce *hole abstraction* $\delta X.M$ which abstracts hole X in term M and creates a term that acts as a context whose hole is X, and we introduce *context application* $M_1 \odot M_2$ which denotes the operation to fill the abstracted hole in M_1 with term M_2. For example, the context $C[\cdot] \equiv (\lambda x.[\cdot] + y)\ 3$ is represented by the term

$$(\delta X.(\lambda x.X + y)\ 3)$$

and the context application term

$$(\delta X.(\lambda x.X + y)\ 3) \odot (x + z)$$

denotes the term obtained by filling the hole in the context with $x+z$. We call a subterm of the form $(\delta X.M_1) \odot M_2$ *fill-redex*, which contracts to the term obtained from M_1 by filling the X-hole in M_1 with M_2. Different from the meta notation $C[x + z]$, context application is a term constructor, which allows us to exploit the features of first-class contexts by combining it with lambda abstraction and lambda application. For example, we can write a term like

$$(\lambda k.k \odot (x + z))\ (\delta X.(\lambda x.X + y)\ 3)$$

which is contracted to the above term.

The goal of this paper is to develop a type system and a reduction system for the lambda calculus extended with the above three-term constructors, i.e., labeled holes,

hole abstraction and context application. The crucial step is the development of a proper mechanism for integrating *variable-capturing* hole-filling and *capture-avoiding* substitution in the lambda calculus. To see the problem, consider the term

$$(\lambda z.(\delta X.(\lambda x.X + y)\ 3) \odot (\mathbf{x} + z))\ \underline{x}$$

where we use different type faces (x, $\mathbf{x}$ and $\underline{x}$) to distinguish different occurrences of variable x to which we should pay attention. The above term has two β-redexes and one fill-redex. Our intention is that the inner $\mathbf{x}$ should be captured by the λx when it is filled in hole X, while the outer $\underline{x}$ is free. The following reduction sequence produces the intended result.

$$\begin{aligned}(\lambda z.(\delta X.(\lambda x.X + y)\ 3) \odot (\mathbf{x} + z))\ \underline{x} &\to (\lambda z.(\lambda x.\mathbf{x} + z + y)\ 3)\ \underline{x}\\ &\to (\lambda z.3 + z + y)\ \underline{x}\\ &\to 3 + \underline{x} + y\end{aligned}$$

However, reducing any of the β-redexes before the fill-redex will result in a different term. If we reduce the inner β-redex before the fill-redex then the binding of inner x will be lost, yielding $\mathbf{x} + \underline{x} + y$. If we reduce the outer β-redex before the fill-redex, then the outer $\underline{x}$ is unintentionally captured by λx, yielding $3 + 3 + y$ or $\mathbf{x} + \underline{x} + y$ depending on the order of the fill-redex and the other β-redex.

To avoid these inconsistencies, we should redefine the scope of lambda binding to reflect the behavior of terms of the form $(\delta X.M_1) \odot M_2$. Suppose there is a λx in M_1 whose scope contains X. Since M_2 is filled in X, the scope of the λx also extends to M_2. This property implies the following two requirements. First, a β-redex containing a hole X cannot be contracted. Secondly, when substituting a term containing x for a free variable in M_2, the λx in M_1 and the corresponding variables in M_1 and M_2 need to be renamed to avoid unwanted capture. In the above example, we should not contract the inner β-redex before hole-filling, and when we contract the outer β-redex before hole-filling, we should rename λx and $\mathbf{x}$ before doing β-substitution. The situation becomes more subtle when we consider a term like

$$(\lambda w.\cdots((\lambda z.w \odot (\mathbf{x} + z))\ \underline{x})\cdots)\ (\delta X.(\lambda x.X + y)\,3).$$

Since w is a variable, simple inspection of term $w \odot (\mathbf{x} + z)$ no longer tells which variables in $\mathbf{x} + z$ should be regarded as bound. However, variable-capture will still occur when the hole abstraction is substituted for w.

Our strategy to solve this problem is to define a type system that tells exactly which variables should be considered bound, and to introduce a refined notion of α-equivalence that reconciles hole-filling and β-substitution.

To tell which variables should be considered bound, we type a hole-abstracted term $\delta X.M$ with a *context type* of the form:

$$[\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\} \rhd \tau_1] \Rightarrow \tau_2,$$

where τ_1 is the type of the abstracted hole, τ_2 is the type of the term that will be produced by filling the hole in the context with a term, and $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$ describes the set of variables being captured when they are filled in the hole X. We call those variables *interface variables*. For example, $\delta X.(\lambda x.X + y)\ 3$ would be typed as $[\{x : \text{int}\} \triangleright \text{int}] \Rightarrow \text{int}$. However, if we simply list the set of actual bound variables surrounding X in $\delta X.M$ as interface variables in its type, then we cannot rename those bound variables. Since β-substitution can only be defined up to renaming of bound variables, this causes a problem in extending substitution to hole-abstracted terms. For example, we cannot rename bound variable x in the term $\delta X.(\lambda x.X + y)\ 3$. It should be noted that the usual "bound variable convention" does not solve the problem. In the lambda calculus, we can simply assume that "all bound variables are different from the free variables" for each β-redex. This is only possible when we can freely rename bound variables. As well known in the theory of lambda calculus, the above condition is not preserved by substitution. Even if we start with a term satisfying the bound variable condition, anomalous terms like the above may appear during β-reduction.

To avoid this problem, we separate actual bound variables in $\delta X.M$ and the corresponding interface variables, and refine hole-filling to an operation that also performs variable renaming. For manipulation of binding structures, Talcott [16] developed a technique to pair a hole with a substitution. We use this approach and annotate a hole X with a *variable renamer* v, which renames interface variables to the corresponding bound variables. We write X^v for the hole X annotated with v. The above context can now be represented as the typed term

$$\delta X.(\lambda a.X^{\{a/x\}} + y)\,3 : [\{x : \text{int}\} \triangleright \text{int}] \Rightarrow \text{int},$$

where x is an interface variable and is renamed to a when it is filled in X. By this separation, bound variable a can be renamed without changing the type of this term. This allows us to achieve a proper integration of hole-filling and β-substitution with terms of the form $\delta X.M$. The semantics of hole-filling is preserved by applying the renamer $\{a/x\}$ to the term to be filled in X. For example, we have the following reduction for the example before.

$$\begin{aligned}(\lambda z.(\delta X.(\lambda a.X^{\{a/x\}} + y)\,3) \odot (\mathbf{x} + z))\,\underline{x} &\to (\lambda z.(\lambda a.\{a/x\}(\mathbf{x} + z) + y)\,3)\ \underline{x} \\ &\equiv (\lambda z.(\lambda a.a + z + y)\,3)\ \underline{x} \\ &\to (\lambda a.a + \underline{x} + y)\,3 \\ &\to 3 + \underline{x} + y\end{aligned}$$

Yet another delicate problem arises when we consider the interaction between substitution and a term of the form $M \odot N$. This construct may bind some variables in N. In order to determine those bound variables, we need to annotate this construct with the set of variables in N that will be bound by forming this term. Since this set must correspond to the set of interface variables of the context term M, a naive attempt would be to annotate the constructor $M \odot N$ with this set. A subterm of the example

term might then be represented as the term $(\delta X.(\lambda a.X^{\{a/x\}} + y)\,3)\odot_{\{x\}}(x+z)$. As we noted earlier, the variable x must be treated as bound variable. This implies that, when combining β-substitution, this variable needs to be renamed. Unfortunately, this is impossible for terms of the form $w\odot_{\{x\}}(x+z)$. Since w is a variable, we cannot rename the corresponding interface variables of the hole-abstracted term that will be substituted later for w. So, again we need to separate the set of interface variables in the type of hole-abstracted term and the set of variables that will be captured when they are filled in the hole of the context. To achieve this, we annotate the constructor for context application with a renamer v and write $M\odot_v N$. The renamer v renames variables in N that are to be bound by hole-filling to the corresponding interface variables in the hole abstracted term. Its effect is obtained by composing it with the renamer of the hole. Now the bound variables in N are independent of the corresponding interface variables, we can perform bound variable renaming. The above example can be correctly represented by the following term:

$$(\delta X.(\lambda a.X^{\{a/x\}} + y)\,3)\odot_{\{x/b\}}(b+z).$$

In this term, both a and b are bound variables, which can be renamed without changing the typing of the term. Again, the semantics of hole-filling is preserved by applying the composition $\{a/x\}\bigstar\{x/b\}(\equiv\{a/b\})$ of renamers $\{a/x\}$ and $\{x/b\}$ to the term to be filled in X. The following is an example of reduction involving renamer applications.

$$\begin{aligned}&(\delta X.(\lambda a.X^{\{a/x\}} + y)\ 3)\odot_{\{x/b\}}(b+z)\\ &\quad\to(\lambda a.(\{a/x\}\bigstar\{x/b\})(b+z)+y)\ 3\equiv(\lambda a.a+z+y)\ 3.\end{aligned}$$

Another slightly more general alternative to $M\odot_v N$ is to make a renamer as a term constructor $[v.N]$ and introduce a new type constructor $[\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}\triangleright\tau]$ for this constructor. We believe that this is also possible. In our system, however, we shall not take this approach, since the only elimination operation would be (the modified version of) the hole-filling and therefore the additional flexibility is not essential in achieving our goal of first-class treatment of contexts.

Based on the strategies outlined above, we have worked out the definition of the type system of the calculus, and its reduction system, and proved that the type system has the subject reduction property and that the reduction system is Church–Rosser.

In the work by Sato et al. [14], a type-theoretical approach similar to ours was taken in order to identify the set of free and bound variables. However, their system does not fully address the problem of mixing such a construct with β-substitution. Their calculus contains a term constructor $e_1[\![e_2]\!]$ whose intuitive meaning is to evaluate e_2 under the bindings provided by the environment e_1. However, the reduction for nested application of this construction is restricted to variables, and does not act on general terms. Because of this restricted treatment, the subtle problem of α-equivalence explained above does not arise in their system.

The careful reader may have noticed that some aspects of contexts can already be represented in the lambda calculus. If one can predetermine the exact order of variables

exported by a context and imported by a term to be filled in the context, then one can represent hole abstractions and context applications simply by functionals as seen in the following encoding scheme. A hole-filling of the form

$$(\delta X.\lambda x_1.\cdots.\lambda x_n.\cdots.X\cdots)\odot M$$

can be represented as a lambda term of the form

$$(\lambda X.\lambda x_1.\cdots.\lambda x_n.\cdots.(Xx_1\cdots x_n)\cdots)\ (\lambda x_1.\lambda x_2.\cdots.\lambda x_n.M).$$

However, such encoding eliminates the ability to bind variables through *names*, and it therefore significantly reduces the benefits of first-class contexts we have advocated in the introduction.

The rest of the paper is organized as follows. In Section 3 we define the context calculus. Section 4 defines the reduction system and proves the subject reduction property and Church–Rosser property of the calculus. Section 5 concludes the paper with the discussion of further investigations. Appendix contains proofs of some of the lemmas.

3. The calculus

We use the following notation for functions. The domain and the codomain of a function f are written as $\mathrm{dom}(f)$ and $\mathrm{cod}(f)$, respectively. We sometimes regard a function as a set of pairs and write $\emptyset$ for the empty function. Let f,g be functions. We write $f;g$ for $f\cup g$ provided that $\mathrm{dom}(f)\cap\mathrm{dom}(g)=\emptyset$. We omit ";" if g is explicitly represented as a set, writing $f\{\ldots\}$ for $f;\{\ldots\}$. The restriction of a function f to the domain D is written as $f|_D$.

The set of types (ranged over by τ) of the calculus is given by the syntax:

$$\tau ::= b \mid \tau\to\tau \mid [\Gamma\triangleright\tau]\Rightarrow\tau,$$

where b ranges over a given set of base types, and Γ ranges over *variable type assignments* each of which is a function from a finite set of variables to types.

We let x range over a countably infinite set of variables; we let X range over a countably infinite set of labeled holes; and we let v range over *variable renamers* each of which is a function from a finite set of variables to variables denoted by $\{y_1/x_1,\ldots,y_n/x_n\}$. Let $v=\{y_1/x_1,\ldots,y_n/x_n\}$ be a renamer. To avoid unnecessary complication, we assume that $\{y_i\mid i=1,\ldots,n\}\cap\{x_i\mid i=1,\ldots,n\}=\emptyset$ or $x_i=y_i$ $(i=1,\ldots,n)$. That is, a renamer changes each name in domain of the renamer to a fresh name, if it is not an identity. A renamer is extended to the set of all variables by letting $v(x)=x$ for all $x\notin\mathrm{dom}(v)$. In what follows, we identify a renamer with its extension. However, we maintain that the domain $\mathrm{dom}(v)$ of a renamer v always means the domain of the original finite function v. The composition $v_1\bigstar v_2$ of two variable renamers v_1 and v_2 is the function v such that $\mathrm{dom}(v)=\mathrm{dom}(v_2)$ and for all $x\in\mathrm{dom}(v_2)$, $v(x)=v_1(v_2(x))$.

$$
\begin{aligned}
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(\lambda x:\tau.M) &= \mathrm{FV}(M) \setminus \{x\} \\
\mathrm{FV}(M_1\ M_2) &= \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) \\
\mathrm{FV}(X^{\nu}) &= \emptyset \\
\mathrm{FV}(\delta X.M) &= \mathrm{FV}(M) \\
\mathrm{FV}(M_1 \odot_{\nu} M_2) &= \mathrm{FV}(M_1) \cup (\mathrm{FV}(M_2) \setminus \mathrm{dom}(\nu)) \\
\mathrm{BV}(x) &= \emptyset \\
\mathrm{BV}(\lambda x:\tau.M) &= \mathrm{BV}(M) \cup \{x\} \\
\mathrm{BV}(M_1\ M_2) &= \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2) \\
\mathrm{BV}(X^{\nu}) &= \emptyset \\
\mathrm{BV}(\delta X.M) &= \mathrm{BV}(M) \\
\mathrm{BV}(M_1 \odot_{\nu} M_2) &= \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2) \cup \mathrm{dom}(\nu)
\end{aligned}
$$

Fig. 1. The sets of free and bound variables.

The set of (unchecked) terms (ranged over by M) of the calculus is given by the syntax:

$$M ::= x \mid \lambda x:\tau.M \mid MM \mid X^{\nu} \mid \delta X.M \mid M \odot_{\nu} M$$

A term $\delta X.M$ binds the hole X in M. The definitions of *bound holes* and *free holes* are given similarly to the usual definition of bound variables and free variables in the ordinary lambda calculus. We write $\mathrm{FH}(M)$ for the set of free holes in M. Since δX is the only binder for holes, this does not create any of the subtle problems we have explained for variables in our calculus, and therefore we can safely assume α-renaming of bound holes just as in α-congruence in the ordinary lambda calculus. In what follows, we regard terms as their α-equivalence classes induced by *bound holes renaming*.

The set of free variables, denoted by $\mathrm{FV}(M)$, and that of bound variables of M, denoted by $\mathrm{BV}(M)$ are given in Fig. 1. These definitions correctly model the effect of context application terms of the form $M_1 \odot_{\nu} M_2$ which binds the variables in $\mathrm{dom}(\nu)$ in M_2.

In addition to the sets of free and bound variables, we need to distinguish three other classes of variables. Let M be a term containing a hole X^{ν}. The variables in $\mathrm{cod}(\nu)$, which we call *free variable candidates*, behave similarly to free variables if they are not abstracted in M; The variables in the set $\mathrm{dom}(\nu)$, which we call *interface variable candidates*, are the source of interface variables. To see the last one, consider a term which contains $M_1 \odot_{\nu} M_2$. The variables in $\mathrm{cod}(\nu)$, which we call *exported variables*, are used to match the variables exported by the context M_1 with bound variables in M_2. The formal definitions of the set $\mathrm{FVC}(M)$ of free variable candidates of M, and the set $\mathrm{IVC}(M)$ of interface variables candidates of M are given in Fig. 2, and the definition of the set $\mathrm{EV}(M)$ of exported variables of M is given in Fig. 3. We define

$$
\begin{aligned}
\mathrm{FVC}(x) &= \emptyset \\
\mathrm{FVC}(\lambda x:\tau.M) &= \mathrm{FVC}(M) \setminus \{x\} \\
\mathrm{FVC}(M_1\ M_2) &= \mathrm{FVC}(M_1) \cup \mathrm{FVC}(M_2) \\
\mathrm{FVC}(X^{\nu}) &= \mathrm{cod}(\nu) \\
\mathrm{FVC}(\delta X.M) &= \mathrm{FVC}(M) \\
\mathrm{FVC}(M_1 \odot_{\nu} M_2) &= \mathrm{FVC}(M_1) \cup (\mathrm{FVC}(M_2) \setminus \mathrm{dom}(\nu)) \\
\mathrm{IVC}(x) &= \emptyset \\
\mathrm{IVC}(\lambda x:\tau.M) &= \mathrm{IVC}(M) \\
\mathrm{IVC}(M_1\ M_2) &= \mathrm{IVC}(M_1) \cup \mathrm{IVC}(M_2) \\
\mathrm{IVC}(X^{\nu}) &= \mathrm{dom}(\nu) \\
\mathrm{IVC}(\delta X.M) &= \mathrm{IVC}(M) \\
\mathrm{IVC}(M_1 \odot_{\nu} M_2) &= \mathrm{IVC}(M_1) \cup \mathrm{IVC}(M_2)
\end{aligned}
$$

Fig. 2. The sets of free variable candidates and interface variable candidates.

$$
\begin{aligned}
\mathrm{EV}(x) &= \emptyset \\
\mathrm{EV}(\lambda x:\tau.M) &= \mathrm{EV}(M) \\
\mathrm{EV}(M_1\ M_2) &= \mathrm{EV}(M_1) \cup \mathrm{EV}(M_2) \\
\mathrm{EV}(X^{\nu}) &= \emptyset \\
\mathrm{EV}(\delta X.M) &= \mathrm{EV}(M) \\
\mathrm{EV}(M_1 \odot_{\nu} M_2) &= \mathrm{EV}(M_1) \cup \mathrm{EV}(M_2) \cup \mathrm{cod}(\nu)
\end{aligned}
$$

Fig. 3. The set of exported variables.

the set $\mathrm{PFV}(M)$ of *potentially free variables* of M as

$$\mathrm{PFV}(M) = \mathrm{FV}(M) \cup \mathrm{FVC}(M).$$

We are now in the position to define the type system of the calculus. Since a term may contain free holes as well as free variables, its type depends not only on types of variables but also on types of free holes. A hole type is determined by a triple $([\Gamma \triangleright \tau], \nu)$ consisting of type τ of a term to be filled, type assignment Γ describing the set of interface variables and their types, and variable renamer ν which is used to keep track of the correspondence between bound variables and interface variables. While Γ describes the set of all abstracted variables, ν describes the set of free variable candidates to be abstracted. We write $\mathrm{Cl}(\{x:\tau\}, ([\Gamma \triangleright \tau], \nu))$ for the triple obtained from $([\Gamma \triangleright \tau], \nu)$ by abstracting x, whose definition is given below.

$$
\begin{aligned}
&\mathrm{Cl}(\{x:\tau\}, ([\Gamma \triangleright \tau], \nu \cup \{x/x'\})) = ([\Gamma\{x':\tau\} \triangleright \tau], \nu) \\
&\mathrm{Cl}(\{x:\tau\}, ([\Gamma \triangleright \tau], \nu)) = ([\Gamma \triangleright \tau], \nu) \quad \text{if } x \notin \mathrm{cod}(\nu).
\end{aligned}
$$

$$\text{(var)}\quad \Gamma, \emptyset \vdash x : \tau \quad \text{if } x \in \mathrm{dom}(\Gamma) \text{ and } \Gamma(x) = \tau$$

$$\text{(abs)}\quad \frac{\Gamma\{x : \tau_1\}, \Delta \vdash M : \tau_2}{\Gamma, \mathrm{Cl}(\{x : \tau_1\}, \Delta) \vdash \lambda x : \tau_1.M : \tau_1 \to \tau_2}$$

$$\text{(app)}\quad \frac{\Gamma, \Delta_1 \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma, \Delta_2 \vdash M_2 : \tau_1}{\Gamma, \Delta_1; \Delta_2 \vdash M_1\ M_2 : \tau_2}$$

$$\text{(hole)}\quad \Gamma, \{X : ([\emptyset \triangleright \tau], \nu)\} \vdash X^{\nu} : \tau \quad \text{if } \mathrm{cod}(\nu) \subseteq \mathrm{dom}(\Gamma)$$

$$\text{(habs)}\quad \frac{\Gamma_1, \Delta\{X : ([\Gamma_2 \triangleright \tau_1], \emptyset)\} \vdash M : \tau_2}{\Gamma_1, \Delta \vdash \delta X.M : [\Gamma_2 \triangleright \tau_1] \Rightarrow \tau_2}$$

$$\text{(fill)}\quad \frac{\begin{array}{l}\Gamma, \Delta_1 \vdash M_1 : [\{x'_1 : \sigma_1, \ldots, x'_n : \sigma_n\} \triangleright \tau_1] \Rightarrow \tau_2 \\ \Gamma\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}, \Delta_2 \vdash M_2 : \tau_1\end{array}}{\Gamma, \Delta_1; \mathrm{Cl}(\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}, \Delta_2) \vdash M_1 \odot_{\{x'_1/x_1, \ldots, x'_n/x_n\}} M_2 : \tau_2}$$
$$\text{if } \mathrm{dom}(\Gamma) \cap \{x'_i | i = 1 \cdots n\} = \emptyset$$

Fig. 4. The type system.

This operation is extended to type assignments as follows:

$$\mathrm{Cl}(\{x_1 : \tau_1, \ldots, x_n : \tau_n\}, ([\Gamma \triangleright \tau], v)) = \mathrm{Cl}(\{x_1 : \tau_1\}, \ldots \mathrm{Cl}(\{x_n : \tau_n\}, ([\Gamma \triangleright \tau], v)) \cdots).$$

A *hole type assignment*, ranged over by Δ, is a finite function which assigns a hole to a triple $([\Gamma \triangleright \tau], v)$ describing the type of the hole, and we call the variables in $\mathrm{dom}(\Gamma)$ *interface variables*. We write $\mathrm{Cl}(\Gamma, \Delta)$ for the hole type assignment $\{X : \mathrm{Cl}(\Gamma, \Delta(X)) | X \in \mathrm{dom}(\Delta)\}$. We write $v\Delta$ for the hole type assignment $\{X : ([\Gamma \triangleright \sigma], v \bigstar v') \mid \{X : ([\Gamma \triangleright \sigma], v')\} \in \Delta\}$.

The type system of the calculus is defined as a proof system to derive a *typing* of the form:

$$\Gamma, \Delta \vdash M : \tau$$

which indicates that term M has type τ under variable type assignment Γ and hole type assignment Δ. The set of typing rules is given in Fig. 4.

Some explanations are in order.

- Rule (hole). Since X is not surrounded by any λ at this moment, the associated type assignment in the hole type assignment is empty, and the set of free variable candidates of X is specified by v. They will be abstracted by the rule (abs) and (fill).
- Rule (abs). Lambda abstracting x not only discharges x from the type hypothesis Γ for the term M, but also extends the set of interface variables for each hole in M with corresponding x'. The later effect is represented by the operation $\mathrm{Cl}(\{x : \tau\}, \Delta)$, which extends each Γ appearing in Δ.

$$\dfrac{\dfrac{\dfrac{\{x : \mathrm{int}\}, \{X : ([\emptyset \triangleright \mathrm{int}], \{x/a\})\} \vdash X^{\{x/a\}} : \mathrm{int}}{\emptyset, \{X : ([\{a : \mathrm{int}\} \triangleright \mathrm{int}], \emptyset)\} \vdash \lambda x : \mathrm{int}.X^{\{x/a\}} : \mathrm{int} \to \mathrm{int}} \quad \emptyset, \emptyset \vdash 3 : \mathrm{int}}{\emptyset, \{X : ([\{a : \mathrm{int}\} \triangleright \mathrm{int}], \emptyset)\} \vdash (\lambda x : \mathrm{int}.X^{\{x/a\}})\ 3 : \mathrm{int}}}{\emptyset, \emptyset \vdash \delta X.(\lambda x : \mathrm{int}.X^{\{x/a\}})\ 3 : [\{a : \mathrm{int}\} \triangleright \mathrm{int}] \Rightarrow \mathrm{int}}$$

Fig. 5. Example of typing derivation.

- Rule (fill). By forming the term $M_1 \odot_{\{x'_1/x_1,\ldots,x'_n/x_n\}} M_2$, each x_i in M_2 becomes bound, and the set of interface variables of each hole in M_2 is extended with it. This property is modeled by discharging each x_i from the typing judgment for M_2 and abstracting it from Δ_2. This rule is similar to the one for a "closure" i.e., a term associated with an explicit substitution, in $\lambda\sigma$-calculus [1].

  Fig. 5 shows an example of typing derivation.

In our calculus, each free hole occurs linearly in a well-typed term. If multiple occurrences of a hole are allowed, then they could have different interface variables. This would considerably complicate the conceptual understanding of contexts as well as the type system. The linearity condition is ensured by the rule (hole), the condition implied by the notation $\Delta_1; \Delta_2$ in rules (app), (fill), and the property that there is no rule for adding redundant hypothesis to Δ. The following lemma is easily shown by induction on the typing derivations.

Lemma 1. *If* $\Gamma, \Delta \vdash M : \tau$, *then* $\mathrm{FH}(M) = \mathrm{dom}(\Delta)$. *Moreover, each free hole appears exactly once in* M.

The following standard properties also hold for this type system, and can be easily shown by induction on the typing derivations.

Lemma 2. *If* $\Gamma, \Delta \vdash M : \tau$, *then* $\mathrm{dom}(\Gamma) \cap \mathrm{BV}(M) = \emptyset$.

Lemma 3. *If* $\Gamma, \Delta \vdash M : \tau$, *then* $\mathrm{PFV}(M) \subseteq \mathrm{dom}(\Gamma)$.

Lemma 4. *If* $\Gamma, \{X_1 : ([\Gamma_1 \triangleright \sigma_1], v_1), \ldots X_n : ([\Gamma_n \triangleright \sigma_n], v_n)\} \vdash M : \tau$, *then* $\mathrm{FVC}(M) = \bigcup_{i=1}^{n} \mathrm{cod}(v_i)$.

Lemma 5. *If* $\Gamma\{x : \sigma\}, \Delta \vdash M : \tau$ *and* $x \notin \mathrm{PFV}(M)$, *then* $\Gamma, \Delta \vdash M : \tau$.

Lemma 6. *If* $\Gamma, \Delta \vdash M : \tau$ *and* $x \notin \mathrm{dom}(\Gamma) \cup \mathrm{BV}(M) \cup \mathrm{EV}(M)$, *then* $\Gamma\{x : \sigma\}, \Delta \vdash M : \tau$.

Lemma 7. *If* $X^{\{x'_1/x_1,\ldots,x'_m/x_m, y'_1/y_1,\ldots,y'_n/y_n\}}$ *occurs in* M, $\{z_i, w_j \mid i = 1 \cdots m,\ j = 1 \cdots n\} \cap \{x'_i, y'_j \mid i = 1 \cdots m,\ j = 1 \cdots n\} = \emptyset$, *and* $\Gamma, \Delta\{X : ([\{x_1 : \sigma_1, \ldots, x_m : \sigma_m\} \triangleright \sigma], \{y'_1/y_1, \ldots, y'_n/y_n\})\} \vdash M : \tau$, *then* $\Gamma, \Delta\{X : ([\{z_1 : \sigma_1, \ldots, z_m : \sigma_m\} \triangleright \sigma], \{y'_1/w_1, \ldots, y'_n/w_n\})\} \vdash M' : \tau$ *where* M' *is obtained from* M *by substituting* $X^{\{x'_1/x_1,\ldots,x'_m/x_m, y'_1/y_1,\ldots,y'_n/y_n\}}$ *for* $X^{\{x'_1/z_1,\ldots,x'_m/z_m, y'_1/w_1,\ldots,y'_n/w_n\}}$.

4. The reduction system

To define the reduction relation, we need to define substitution and hole-filling operations. In the ordinary lambda calculus, substitution can be defined modulo α-congruence, which allows us to simply assume that unwanted variable capture will not happen. In our calculus, since we have not yet obtained α-congruence, we need at first to define substitution as an operation on syntactic terms (not on equivalence class).

We write $\{M'/x\}M$ for the term obtained by substituting M' for any free occurrence of x in M. The following lemma shows that substitution preserves typing under a strong variable hygiene conditions.

Lemma 8. *If* $\Gamma, \emptyset \vdash M_0 : \sigma$, $\Gamma\{x:\sigma\}, \Delta \vdash M : \tau$, $\mathrm{BV}(M) \cap (\mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ *and* $x \notin \mathrm{FVC}(M)$, *then* $\Gamma, \Delta \vdash \{M_0/x\}M : \tau$.

The proof is deferred to the appendix. As in the standard definition of substitution, we have the following composition lemma:

Lemma 9. $\{M_1/x\}(\{M_2/y\}M_3) \equiv \{(\{M_1/x\}M_2)/y\}(\{M_1/x\}M_3)$ *where* $y \notin \mathrm{PFV}(M_1)$ *and* $x \not\equiv y$.

As we have explained earlier, hole-filling involves application of the variable renamer associated with the hole to the term being filled. To define hole-filling, we extend a variable renamer v to a function $\overline{v}$ on terms as follows:

$$\begin{aligned}
\overline{v}(x) &= v(x) \\
\overline{v}(\lambda x:\tau.M) &= \lambda x:\tau.(\overline{v|_{(\mathrm{dom}(v)\setminus\{x\})}})M \\
\overline{v}(M_1\ M_2) &= \overline{v}M_1\ \overline{v}M_2 \\
\overline{v}(X^{v'}) &= X^{v \star v'} \\
\overline{v}(\delta X.M) &= \delta X.\overline{v}M \\
\overline{v}(M_1 \odot_{v'} M_2) &= \overline{v}M_1 \odot_{v'} (\overline{v|_{(\mathrm{dom}(v)\setminus\mathrm{dom}(v'))}})M_2
\end{aligned}$$

We have the following renaming lemma, whose proof is deferred to the appendix.

Lemma 10. *If* $\Gamma\{x:\sigma\}, \Delta \vdash M : \tau$ *and* $x' \notin \mathrm{dom}(\Gamma) \cup \mathrm{BV}(M) \cup \mathrm{IVC}(M) \cup \mathrm{EV}(M)$, *then* $\Gamma\{x':\sigma\}, \{x'/x\}\Delta \vdash \overline{\{x'/x\}}M : \tau$.

Hole-filling is defined as a combination of variable renamer and substitution. We write $M[M'/X]$ for the term obtained from M by syntactically substituting the term $\overline{v}(M')$ for X in M where v is the variable renamer associated with X. Its definition is obtained by simply extending the following clauses according to the structure of M.

$$(X^v)[M'/X] = \overline{v}M' \quad x[M'/X] = x$$

From this definition and the property of typing, it is easily seen that if $\Gamma, \Delta \vdash M : \tau$ and $X \notin \mathrm{dom}(\Delta)$, then $M[M_0/X] \equiv M$. The following lemma shows that hole-filling preserves the typing.

Lemma 11. *If* $\Gamma_1;\Gamma_2\{x'_1:\sigma_1,\ldots,x'_n:\sigma_n\},\Delta_0\vdash M_0:\sigma$,
$\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta\{X:([\Gamma_2\triangleright\sigma],\{x_1/x'_1,\ldots,x_n/x'_n\})\}\vdash M:\tau$,
$(\mathrm{BV}(M)\cup\{x_i\mid i=1\cdots n\})\cap(\mathrm{IVC}(M_0)\cup\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ *and* $\mathrm{BV}(M)\cap(\mathrm{dom}(\Gamma_2)\cup\{x'_i\mid i=1\cdots n\})=\emptyset$, *then*
$\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta;\mathrm{Cl}(\Gamma_2,\{x_1/x'_1,\ldots,x_n/x'_n\}\Delta_0)\vdash M[M_0/X]:\tau$.

The proof is deferred to the appendix.

The following is the composition lemma for the hole-filling, where $\mathrm{IVC}_X(M)$ denotes the domain of the variable renamer on the shoulder of hole X in M.

Lemma 12. *If* $X\not\equiv Y, Y\notin\mathrm{FH}(M_1)$ *and* $\mathrm{IVC}_X(M_3 f)\cap(\mathrm{PFV}(M_1)\setminus\mathrm{IVC}_Y(M_2))=\emptyset$, *then* $(M_3[M_2/X])[M_1/Y]\equiv M_3[M_1/Y][M_2[M_1/Y]/X]$.

Proof. If $\mathrm{dom}(v_1)\cap(\mathrm{PFV}(M)\setminus\mathrm{dom}(v_2))=\emptyset$, then we have $\overline{(v_1\bigstar v_2)}M\equiv\overline{v_1}(\overline{v_2}M))$. □

The notion of α-congruence in our calculus is now defined as the congruence relation on the set of well-typed terms generated by the following two axioms:

- $\lambda x:\tau.M\equiv_\alpha\lambda y:\tau.\overline{\{y/x\}}M$ if $y\notin\mathrm{BV}(M)\cup\mathrm{PFV}(M)\cup\mathrm{IVC}(M)\cup\mathrm{EV}(M)$
- $M_1\odot_{\{x'_1/x_1,\ldots,x'_n/x_n\}}M_2\equiv_\alpha M_1\odot_{\{x'_1/y_1,\ldots,x'_n/y_n\}}\overline{\{y_1/x_1,\ldots,y_n/x_n\}}M_2$
  if each $y_i\notin\mathrm{BV}(M_2)\cup\mathrm{PFV}(M_2)\cup\mathrm{IVC}(M_2)\cup\mathrm{EV}(M)$

The following lemma shows that α-renaming preserves typing, which is proved by induction on the derivation of M using lemma 10.

Lemma 13. *If* $\Gamma,\Delta\vdash M:\tau$ *and* $M\equiv_\alpha M'$, *then* $\Gamma,\Delta\vdash M':\tau$.

α-congruence allows us to rename bound variables whenever it is necessary. In what follows, we assume the following variable convention for our calculus:

bound variables are all distinct and the set of bound variables has no intersection with the set of interface variable candidates, the set of potentially free variables, and the set of exported variables.

Under this variable convention, the reduction axioms of our calculus are given as follows:

$$(\beta)\ (\lambda x:\sigma.M_1)\ M_2\underset{\beta}{\rightarrow}\{M_2/x\}M_1\ \text{ if }\ \mathrm{FH}(M_1)=\mathrm{FH}(M_2)=\emptyset$$

$$(\mathit{fill})\ (\delta X.M_1)\odot_v M_2\underset{\mathit{fill}}{\rightarrow}(M_1[X^v/X])[M_2/X].$$

In the axiom (β), the restriction $\mathrm{FH}(M_2)=\emptyset$ is needed to ensure that each free hole occurs linearly. The restriction $\mathrm{FH}(M_1)=\emptyset$ is needed to maintain the binding generated by λx for the holes in M_1. Since in our calculus contexts are represented not by terms with free holes but by hole abstracted terms, this does not restrict first-class treatment of contexts.

$$\text{(var)}\ \ x \Longrightarrow x \qquad\qquad \text{(abs)}\ \ \frac{M \Longrightarrow M'}{\lambda x:\tau.M \Longrightarrow \lambda x:\tau.M'}$$

$$\text{(app)}\ \ \frac{M_1 \Longrightarrow M_1' \quad M_2 \Longrightarrow M_2'}{M_1\ M_2 \Longrightarrow M_1'\ M_2'}$$

$$\text{(betred)}\ \ \frac{M_1 \Longrightarrow M_1' \quad M_2 \Longrightarrow M_2'}{(\lambda x:\tau.M_1)\ M_2 \Longrightarrow \{M_2'/x\}\ M_1'}\ \ \text{if } \mathrm{FH}(M_1) = \mathrm{FH}(M_2) = \emptyset$$

$$\text{(hole)}\ \ X^{\nu} \Longrightarrow X^{\nu} \qquad\qquad \text{(habs)}\ \ \frac{M \Longrightarrow M'}{\delta X.M \Longrightarrow \delta X.M'}$$

$$\text{(fill)}\ \ \frac{M_1 \Longrightarrow M_1' \quad M_2 \Longrightarrow M_2'}{M_1 \odot_{\nu} M_2 \Longrightarrow M_1' \odot_{\nu} M_2'}$$

$$\text{(filred)}\ \ \frac{M_1 \Longrightarrow M_1' \quad M_2 \Longrightarrow M_2'}{(\delta X.M_1) \odot_{\nu} M_2 \Longrightarrow (M_1'[X^{\nu}/X])[M_2'/X]}$$

Fig. 6. Definition of the parallel reduction.

The one-step reduction relation $M \to N$ is defined on the set of *well-typed terms* as: $M \to M'$ iff M is well typed and M' is obtained by applying one of the two reduction axioms to some subterm of M. We write $M \xrightarrow{*} M'$ for the reflexive, transitive closure of $\to$.

For this reduction, we have the following desired results.

Theorem 14 (subject reduction). *If* $\Gamma, \Delta \vdash M : \tau$ *and* $M \xrightarrow{*} M'$, *then* $\Gamma, \Delta \vdash M' : \tau$.

Proof. This is a direct consequence of Lemmas 7, 8, 11 and 13. □

Theorem 15 (confluence). *For any well typed term* M, *if* $M \xrightarrow{*} M_1$ *and* $M \xrightarrow{*} M_2$, *then there is some* M_3 *such that* $M_1 \xrightarrow{*} M_3$ *and* $M_2 \xrightarrow{*} M_3$.

The proof is by using the technique of parallel reduction due to Tait and Martin-Löf. The parallel reduction relation of our calculus, written $\rightrightarrows$, is given in Fig. 6.

From this definition, it is easily seen that the transitive closure of the parallel reduction coincides with the reduction relation of the calculus ($\xrightarrow{*}$). To prove the theorem, it is therefore sufficient to prove the diamond property of $\rightrightarrows$. To show this, we follow Takahashi [15] and prove the following stronger property.

Lemma 16. *For any well-typed term* M, *there exists a term* M^* *such that if* $M \rightrightarrows M'$, *then* $M' \rightrightarrows M^*$.

In the lemma above, M^* denotes the term obtained from M by parallel reducing all the possible redexes of M, whose definition is given Fig. 7.

$$
\begin{aligned}
x^* &= x \\
(\lambda x:\tau.M)^* &= \lambda x:\tau.M^* \\
(M_1\ M_2)^* &= M_1^*\ M_2^* \ \text{ if } \mathrm{FH}(M_1)\neq\emptyset \text{ or } \mathrm{FH}(M_2)\neq\emptyset \text{ or} \\
&\qquad\qquad M_1 \text{ is not a lambda abstraction} \\
((\lambda x:\tau.M_1)\ M_2)^* &= \{M_2^*/x\}M_1^* \text{ if } \ \mathrm{FH}(M_1)=\mathrm{FH}(M_2)=\emptyset \\
(X^\nu)^* &= X^\nu \\
(\delta X.M)^* &= \delta X.M^* \\
(M_1\odot_\nu M_2)^* &= M_1^*\odot_\nu M_2^* \ \text{ if } M_1 \text{ is not a hole abstraction} \\
((\delta X.M_1)\odot_\nu M_2)^* &= (M_1^*[X^\nu/X])[M_2^*/X]
\end{aligned}
$$

Fig. 7. Definition of M^*.

The proof of Lemma 16 is by induction on the derivation of $\Gamma,\Delta\vdash M \rightrightarrows M':\tau$ using the following lemmas:

Lemma 17. *If $M \rightrightarrows M'$, $M_0 \rightrightarrows M_0'$, then $\{M_0/x\}M \rightrightarrows \{M_0'/x\}M'$ for any terms M, M_0 such that $\Gamma\{x:\sigma\},\emptyset\vdash M:\tau$ and $\Gamma,\emptyset\vdash M_0:\sigma$ for some $\Gamma,\Delta,\tau,\sigma$.*

Proof. We proceed by induction on the derivation of $M \rightrightarrows M'$. Here we only show the cases (betred) and (filred).

Case (betred) $\dfrac{M_1 \rightrightarrows M_1' \quad M_2 \rightrightarrows M_2'}{(\lambda y:\tau.M_1)\ M_2 \rightrightarrows \{M_2'/y\}\ M_1'}$

($\mathrm{FH}(M_1)=\mathrm{FH}(M_2)=\emptyset$ and $y\not\equiv x$).
By the induction hypothesis,
$\{M_0/x\}M_1 \rightrightarrows \{M_0'/x\}M_1'$ and $\{M_0/x\}M_2 \rightrightarrows \{M_0'/x\}M_2'$.
Therefore, by the rule(betred),
$(\lambda y:\tau.\{M_0/x\}M_1)\ \{M_0/x\}M_2 \rightrightarrows \{\{M_0'/x\}M_2'/y\}\ \{M_0'/x\}M_1'$. The rest of this case is by Lemma 9.

Case (filred) $\dfrac{M_1 \rightrightarrows M_1' \quad M_2 \rightrightarrows M_2'}{(\delta X.M_1)\odot_\nu M_2 \rightrightarrows (M_1'[X^\nu/X])[M_2'/X]}$.

By the induction hypothesis,
$\{M_0/x\}M_1 \rightrightarrows \{M_0'/x\}M_1'$ and $\{M_0/x\}M_2 \rightrightarrows \{M_0'/x\}M_2'$.
Therefore, by the rule(filred),
$(\delta X.\{M_0/x\}M_1)\odot_\nu \{M_0/x\}M_2 \rightrightarrows (\{\{M_0'/x\}M_1'\}[X^\nu/X])[\{M_0'/x\}M_2'/X]$. We can assume $x\notin \mathrm{dom}(\nu)$ by the hygiene condition. Then,
$(\{\{M_0'/x\}M_1'\}[X^\nu/X])[\{M_0'/x\}M_2'/X] \equiv \{M_0'/x\}(M_1'[X^\nu/X])[M_2'/X]$. □

Lemma 18. *If $M \rightrightarrows M'$ and $\{x,x'\}\cap \mathrm{BV}(M)=\emptyset$, then $\overline{\{x'/x\}}M \rightrightarrows \overline{\{x'/x\}}M'$.*

Proof. We proceed by induction on the derivation of $M \rightrightarrows M'$. We only show the crucial case (filred).

$$\textit{Case} \quad \text{(filred)} \quad \frac{M_1 \rightrightarrows M_1' \quad M_2 \rightrightarrows M_2'}{(\delta X.M_1)\odot_v M_2 \rightrightarrows (M_1'[X^v/X])[M_2'/X]}.$$

By the induction hypothesis,
$\overline{\{x'/x\}}M_1 \rightrightarrows \overline{\{x'/x\}}M_1'$ and $\overline{\{x'/x\}}M_2 \rightrightarrows \overline{\{x'/x\}}M_2'$.
Then $(\delta X.\overline{\{x'/x\}}M_1)\odot_v \overline{\{x'/x\}}M_2 \rightrightarrows ((\overline{\{x'/x\}}M_1')[X^v/X])[\overline{\{x'/x\}}M_2'/X]$ by the rule (filred). Since $\{x,x'\}\cap \mathrm{BV}((\delta X.M_1)\odot_v M_2)=\emptyset$, $\mathrm{dom}(v)\cap\{x,x'\}=\emptyset$. Therefore, $((\overline{\{x'/x\}}M_1')[X^v/X])[\overline{\{x'/x\}}M_2'/X] \equiv \overline{\{x'/x\}}((M_1'[X^v/X])[M_2'/X])$. □

Lemma 19. *If $M \rightrightarrows M'$ and $M_0 \rightrightarrows M_0'$, then $M[M_0/X] \rightrightarrows M'[M_0'/X]$ for any terms M, M_0 such that $\Gamma\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}, \Delta\{X:([\Gamma'\triangleright\sigma],\{x_1/x_1',\ldots,x_n/x_n'\})\} \vdash M:\tau$ and $\Gamma;\Gamma'\{x_1':\sigma_1,\ldots,x_n':\sigma_n\},\Delta_0 \vdash M_0:\sigma$ for some $\Gamma,\Gamma',\Delta,\Delta_0,\tau,\sigma,x_1,\ldots,x_n,x_1',\ldots,x_n'$.*

Proof. We proceed by induction on the derivation of $M \rightrightarrows M'$. Here we only show the cases (hole) and (filred).
Case (hole) $X^v \rightrightarrows X^v$. By repeated application of Lemma 18.

$$\textit{Case} \quad \text{(filred)} \quad \frac{M_1 \rightrightarrows M_1' \quad M_2 \rightrightarrows M_2'}{(\delta Y.M_1)\odot_v M_2 \rightrightarrows (M_1'[Y^v/Y])[M_2'/Y]} \quad (Y \not\equiv X).$$

Suppose $X \in \mathrm{FH}(M_1)$. Then, $M_2[M_0/X] \equiv M_2$ and $M_2'[M_0'/X] \equiv M_2'$. By the induction hypothesis, $M_1[M_0/X] \rightrightarrows M_1'[M_0'/X]$. Therefore, by the rule(filred), $(\delta Y.M_1[M_0/X])\odot_v M_2 \rightrightarrows ((M_1'[M_0'/X])[Y^v/Y])[M_2'/Y]$. The rest of this subcase is by $(M_1'[M_0'/X])[Y^v/Y] \equiv (M_1'[Y^v/Y])[M_0'/X]$. Suppose $X \in \mathrm{FH}(M_2)$. Then, $M_1[M_0/X] \equiv M_1$ and $M_1'[M_0'/X] \equiv M_1'$. By the induction hypothesis, $M_2[M_0/X] \rightrightarrows M_2'[M_0'/X]$. Therefore, by the rule (filred),
$(\delta Y.M_1)\odot_v M_2[M_0/X] \rightrightarrows (M_1'[Y^v/Y])[M_2'[M_0'/X]/Y]$. By Lemmas 2 and 3,
$\mathrm{IVC}_Y(M_1)\cap(\mathrm{PFV}(M_0)\backslash\mathrm{IVC}_X(M_2))=\emptyset$. Therefore, by Lemma 12,
$(M_1'[Y^v/Y])[M_2'[M_0'/X]/Y] \equiv ((M_1'[Y^v/Y])[M_2'/Y])[M_0'/X]$.

This completes the proof of Theorem 15. □

5. Conclusions

We have developed a typed calculus for contexts. In this calculus, contexts and lambda terms share the same set of variables and can be freely mixed (as far as they type-check). This allows us to treat contexts truly as first-class values. However, a straightforward mixture of β-reduction and fill-reduction results in an inconsistent system. We have solved the problem by developing a type system that precisely specifies the variable-capturing nature of contexts. The resulting typed calculus enjoys the

subject reduction property and Church–Rosser property. We believe that the typed context calculus presented here will serve as a type theoretical basis for developing a programming language with advanced features for manipulation of open terms. There are a number of interesting topics that merit further investigation. We briefly discuss some of them below.

Integration with explicit substitution: In our calculus, β-contraction is restricted to those redexes that do not contain free holes. While this does not restrict first-class treatment of contexts, removing this restriction will make the reduction system slightly more general. As we have noted earlier, one reason for this restriction is that if we contract a β-redex containing a free hole, then the binding through the hole will be lost. One way of solving this problem would be to integrate our calculus with $\lambda\sigma$-calculus of Abadi et al. [1], and to generalize variable renamers to explicit substitutions. Dowek et al. [3] considered a calculus containing holes and *grafting*, which roughly corresponds to hole-filling, and developed a technique to mingle capture-avoiding substitution with grafting by encoding them in a calculus of explicit substitution using de Bruijn notation. Although their calculus does not contain a term constructor for context application and therefore their technique is not directly applicable to our calculus, we believe that it is possible to extend their technique for our calculus by translating all the machinery we have developed for our calculus into de Bruijn notation. However, such translation would significantly decrease the flexibility of access to exported variables by names. It should also be noted that the notion of de Bruijn indexes presupposes α-equivalence on terms, and therefore defining the context calculus using de Bruijn notation requires the mechanisms (or something similar to those) for obtaining α-equivalence we have developed in this paper.

Programming languages with contexts: Our motivation is to provide a basis for developing a programming language with the feature of first-class contexts. The context calculus we have worked out in this article guarantees that we can have such a typed language with first-class contexts. In order to develop an actual programming language, however, we need to develop a realistic evaluation strategy for the calculus. Our preliminary investigation shows that the usual call-by-value evaluation strategy using closures can be extended to our calculus. A more challenging topic is to develop a polymorphic-type system and a type inference algorithm for our calculus, which will enable us to develop an ML-style programming language with the feature of contexts we have advocated. One crucial issue is the flexible treatment of context types. In the current definition, the constructor $\_\odot_{\{x_1/x'_1,\ldots,x_n/x'_n\}}\_$ is annotated with a variable renamer. This reduces the flexibility of the calculus. A better approach would be to refine the type system so that if $\Gamma \subseteq \Gamma'$, then a context of type $[\Gamma' \triangleright \tau] \Rightarrow \sigma$ can be used whenever a context of type $[\Gamma \triangleright \tau] \Rightarrow \sigma$ is allowed. One of the authors has recently developed an ML-style language with first-class contexts [5] where an ML-style polymorphic-type system, a call-by-value operational semantics and a type inference algorithm are given.

Relationship with formula-as-type notion: It is intuitively clear that a context represented as a term in our calculus has constructive meaning. An important question is to characterize this intuition formally in the sense of Curry–Howard isomorphism

[7]. This would lead us to a new form of proof normalization process corresponding to our fill-reduction. Since the context calculus is Church–Rosser, it should be possible to develop a proof system that is conservative over the conventional intuitionistic logic and supports a proof normalization process corresponding to fill-reduction. The authors recently noticed that there is an intriguing similarity between the proof system of typings in the context calculus and Joshi and Kulick's partial proof manipulation system [8] which is used to represent linguistic information. Another relevant system is Herbelin's lambda calculus isomorphic to a variant of sequent calculus, where proofs of certain sequents are interpreted by applicative contexts [6]. These results suggest some interesting connections between context calculus and proof systems.

Acknowledgements

The authors thank Pierre-Louis Curien, Laurent Dami, Yasuhiko Minamide, Didier Rémy, Masahiko Sato and anonymous referees for their careful reading of a draft of this paper and numerous useful comments. The second author also thanks Shinn-Der Lee, and Dan Friedman for insightful discussions on contexts.

Appendix A. Proofs

Lemma 8. *If* $\Gamma, \emptyset \vdash M_0 : \sigma$, $\Gamma\{x:\sigma\}, \Delta \vdash M : \tau$, $\mathrm{BV}(M) \cap (\mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ *and* $x \notin \mathrm{FVC}(M)$, *then* $\Gamma, \Delta \vdash \{M_0/x\}M : \tau$.

Proof. We proceed by inuction on the derivation of $\Gamma\{x:\sigma\}, \Delta \vdash M : \tau$.

Case (var) $\Gamma\{x:\sigma\}, \emptyset \vdash x : \sigma$. Trivial.

Case (var) $\Gamma\{x:\sigma\}, \emptyset \vdash y : \tau \quad (y \not\equiv x)$. Since $y \not\equiv x$, also the judgment $\Gamma, \emptyset \vdash y : \tau$ is derivable.

Case (abs) $\dfrac{\Gamma\{x:\sigma\}\{y:\tau_1\}, \Delta' \vdash M_1 : \tau_2}{\Gamma\{x:\sigma\}, \Delta \vdash \lambda y : \tau_1.M_1 : \tau_1 \to \tau_2}$
$(\Delta = \mathrm{Cl}(\{y:\tau_1\}, \Delta')$ and $y \not\equiv x)$.

Suppose $\mathrm{BV}(\lambda y:\tau_1.M_1) \cap (\mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ and $x \notin \mathrm{FVC}(\lambda y:\tau_1.M_1)$. Since $\mathrm{BV}(\lambda y:\tau_1.M_1) = \mathrm{BV}(M_1) \cup \{y\}$, $\mathrm{FVC}(\lambda y:\tau_1.M_1) = \mathrm{FVC}(M_1)\backslash\{y\}$ and $y \not\equiv x$, the conditions $\mathrm{BV}(M_1) \cap (\mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ and $x \notin \mathrm{FVC}(M_1)$ hold, and $\Gamma\{y:\tau_1\}, \emptyset \vdash M_0 : \sigma$ by Lemma 6. Therefore, $\Gamma\{y:\tau_1\}, \Delta \vdash \{M_0/x\}M_1 : \tau_2$ by the induction hypothesis. The rest of this case is by the rule (abs).

Case (app) $\dfrac{\Gamma\{x:\sigma\}, \Delta_1 \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma\{x:\sigma\}, \Delta_2 \vdash M_2 : \tau_1}{\Gamma\{x:\sigma\}, \Delta_1; \Delta_2 \vdash M_1\ M_2 : \tau_2}$.

Suppose $\mathrm{BV}(M_1\ M_2) \cap (\mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ and $x \notin \mathrm{FVC}(M_1\ M_2)$. Since $\mathrm{BV}(M_1\ M_2) = \mathrm{BV}(M_1) \cup \mathrm{BV}(M_2)$ and $\mathrm{FVC}(M_1\ M_2) = \mathrm{FVC}(M_1) \cup \mathrm{FVC}(M_2)$, also

$\mathrm{BV}(M_1)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$, $x\notin\mathrm{FVC}(M_1)$, $\mathrm{BV}(M_2)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ and $x\notin\mathrm{FVC}(M_2)$. Therefore, by the induction hypothesis, we have $\Gamma,\Delta_1\vdash\{M_0/x\}M_1:\tau_1\to\tau_2$ and $\Gamma,\Delta_2\vdash\{M_0/x\}M_2:\tau_1$. The rest is by the rule (app).

Case (hole) $\Gamma\{x:\sigma\},\{X:([\emptyset\triangleright\tau],v)\}\vdash X^v:\tau$ $(x\notin\mathrm{cod}(v))$. Since $x\notin\mathrm{PFV}(X^v)$, we have $\Gamma,\{X:([\emptyset\triangleright\tau],v)\}\vdash X^v:\tau$ by Lemma 5.

$$\textit{Case}\ (\text{habs})\quad\frac{\Gamma_1\{x:\sigma\},\Delta\{X:([\Gamma_2\triangleright\tau_1],\emptyset)\}\vdash M_1:\tau_2}{\Gamma_1\{x:\sigma\},\Delta\vdash\delta X.M_1:[\Gamma_2\triangleright\tau_1]\Rightarrow\tau_2}.$$

Suppose $\mathrm{BV}(\delta X.M_1)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ and $x\notin\mathrm{FVC}(\delta X.M_1)$. By the definition of BV and FVC, $\mathrm{BV}(M_1)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ and $x\notin\mathrm{FVC}(M_1)$. Therefore, by the induction hypothesis, we obtain
$\Gamma_1,\Delta\{X:([\Gamma_2\triangleright\tau_1],\emptyset)\}\vdash\{M_0/x\}M_1:\tau_2$. The rest is by the rule (habs).

$$\textit{Case}\quad(\text{fill})\quad\frac{\begin{array}{l}\Gamma\{x:\sigma\},\Delta_1\vdash M_1:[\{x_1':\sigma_1,\ldots,x_n':\sigma_n\}\triangleright\tau_1]\Rightarrow\tau_2\\ \Gamma\{x:\sigma\}\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_2\vdash M_2:\tau_1\end{array}}{\Gamma\{x:\sigma\},\Delta_1;\mathrm{Cl}(\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_2)\vdash M_1\odot_v M_2:\tau_2}$$

$(v=\{x_1'/x_1,\ldots,x_n'/x_n\}$ and $(\mathrm{dom}(\Gamma)\cup\{x\})\cap\{x_i'|i=1\cdots n\}=\emptyset)$.

Suppose $\mathrm{BV}(M_1\odot_v M_2)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ and $x\notin\mathrm{FVC}(M_1\odot_v M_2)$. By the definition of BV, EV and FVC, and by $x\not\equiv x_i$, we have $\{x_i\mid i=1\cdots n\}\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$, $\mathrm{BV}(M_1)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset, x\notin\mathrm{FVC}(M_1)$ $\mathrm{BV}(M_2)\cap(\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$, and $x\notin\mathrm{FVC}(M_2)$. Therefore, by the induction hypothesis applied to
$\Gamma\{x:\sigma\},\Delta_1\vdash M_1:[\{x_1':\sigma_1,\ldots,x_n':\sigma_n\}\triangleright\tau_1]\Rightarrow\tau_2$,
we have $\Gamma,\Delta_1\vdash\{M_0/x\}M_1:[\{x_1':\sigma_1,\ldots,x_n':\sigma_n\}\triangleright\tau_1]\Rightarrow\tau_2$. By Lemma 6, we obtain $\Gamma\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_0\vdash M_0:\sigma$. Therefore, by the induction hypothesis applied to $\Gamma\{x:\sigma\}\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_2\vdash M_2:\tau_1$, we have $\Gamma\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_2\vdash\{M_0/x\}M_2:\tau_2$. The rest of this case is shown by the rule (fill). □

Lemma 10. *If* $\Gamma\{x:\sigma\},\Delta\vdash M:\tau$ *and* $x'\notin\mathrm{dom}(\Gamma)\cup\mathrm{BV}(M)\cup\mathrm{IVC}(M)\cup\mathrm{EV}(M)$, *then* $\Gamma\{x':\sigma\},\{x'/x\}\Delta\vdash\overline{\{x'/x\}}M:\tau$.

Proof. We proceed by induction on the derivation of $\Gamma\{x:\sigma\},\Delta\vdash M:\tau$. The case of (var) is similar to Lemma 8.

$$\textit{Case}\quad(\text{abs})\quad\frac{\Gamma\{x:\sigma\}\{y:\tau_1\},\Delta'\vdash M_1:\tau_2}{\Gamma\{x:\sigma\},\Delta\vdash\lambda y:\tau_1.M_1:\tau_1\to\tau_2}\quad(\Delta=\mathrm{Cl}(\{y:\tau_1\},\Delta')).$$

Suppose $x'\notin\mathrm{dom}(\Gamma)\cup\mathrm{BV}(\lambda y:\tau_1.M_1)\cup\mathrm{IVC}(\lambda y:\tau_1.M_1)\cup\mathrm{EV}(\lambda y:\tau_1.M_1)$. Since $\mathrm{BV}(\lambda y:\tau_1.M_1)=\mathrm{BV}(M_1)\cup\{y\}$, $\mathrm{IVC}(\lambda y:\tau_1.M_1)=\mathrm{IVC}(M_1)$ and $\mathrm{EV}(\lambda y:\tau_1.M_1)=\mathrm{EV}(M_1)$, we have $\Gamma\{x':\sigma\}\{y:\tau_1\},\{x'/x\}\Delta'\vdash\overline{\{x'/x\}}M_1:\tau_2$ by the induction hypothesis. Then, we obtain
$\Gamma\{x':\sigma\},\mathrm{Cl}(\{y:\sigma\},\{x'/x\}\Delta')\vdash\lambda y:\tau_1.\overline{\{x'/x\}}M_1:\tau$ by the rule (abs). Since $y\not\equiv x$ and

$y \not\equiv x'$, $\{x'/x\}(\mathrm{Cl}(\{y:\sigma\}, \Delta')) = \mathrm{Cl}(\{y:\sigma\}, \{x'/x\}\Delta')$ and $\overline{\{x'/x\}}(\lambda y:\tau_1.M_1) \equiv \lambda y:\tau_1.\overline{\{x'/x\}}M_1$.

$$\textit{Case}\ \ (\text{app})\ \ \frac{\Gamma\{x:\sigma\}, \Delta_1 \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma\{x:\sigma\}, \Delta_2 \vdash M_2 : \tau_1}{\Gamma\{x:\sigma\}, \Delta \vdash M_1\ M_2 : \tau_2} \quad (\Delta = \Delta_1; \Delta_2).$$

Suppose $x' \notin \mathrm{dom}(\Gamma) \cup \mathrm{BV}(M_1\ M_2) \cup \mathrm{IVC}(M_1\ M_2) \cup \mathrm{EV}(M_1\ M_2)$. By the definition of BV, IVC and EV, we can apply the induction hypothesis. The rest is by the rule (app).

Case (hole) $\Gamma\{x:\sigma\}, \{X:([\emptyset \triangleright \tau], v)\} \vdash X^v : \tau$ $(x \notin \mathrm{cod}(v))$. Trivial.

Case (hole) $\Gamma\{x:\sigma\}, \{X:([\emptyset \triangleright \tau], v\{x/w\})\} \vdash X^{v\{x/w\}} : \tau$. Since $x' \notin \mathrm{IVC}(X^{v\{x/w\}}) = \mathrm{dom}(v) \cup \{w\}$, we have $\Gamma\{x':\sigma\}, \{X:([\emptyset \triangleright \tau], v\{x'/w\})\} \vdash X^{v\{x'/w\}} : \tau$ by the well-formedness of the variable renamer.

$$\textit{Case}\ (\text{habs})\ \ \frac{\Gamma_1\{x:\sigma\}, \Delta\{X:([\Gamma_2 \triangleright \tau_1], \emptyset)\} \vdash M_1 : \tau_2}{\Gamma_1\{x:\sigma\}, \Delta \vdash \delta X.M_1 : [\Gamma_2 \triangleright \tau_1] \Rightarrow \tau_2}.$$

Suppose $x' \notin \mathrm{dom}(\Gamma_1) \cup \mathrm{BV}(\delta X.M_1) \cup \mathrm{IVC}(\delta X.M_1) \cup \mathrm{EV}(\delta X.M_1)$. By the definition of BV, IVC and EV, we can apply the induction pothesis; we have $\Gamma_1\{x':\sigma\}(\Delta\{X:([\Gamma_2 \triangleright \tau_1], \emptyset\}) \vdash \overline{\{x'/x\}}M_1 : \tau_2$. The rest is by the rule (habs).

$$\textit{Case}\ \ (\text{fill})\ \ \frac{\begin{array}{l}\Gamma\{x:\sigma\}, \Delta_1 \vdash M_1 : [\{x'_1:\sigma_1, \ldots, x'_n:\sigma_n\} \triangleright \tau_1] \Rightarrow \tau_2 \\ \Gamma\{x:\sigma\}\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \Delta_2 \vdash M_2 : \tau_1\end{array}}{\Gamma\{x:\sigma\}, \Delta_1; \mathrm{Cl}(\{x_i:\sigma_i\}, \Delta_2) \vdash M_1 \odot_v M_2 : \tau_2}$$

$(v = \{x'_1/x_1, \ldots, x'_n/x_n\}$ and $(\mathrm{dom}(\Gamma) \cup \{x\}) \cap \{x'_i | i = 1 \cdots n\} = \emptyset)$.

Suppose $x' \notin \mathrm{dom}(\Gamma) \cup \mathrm{BV}(M_1 \odot_v M_2) \cup \mathrm{IVC}(M_1 \odot_v M_2) \cup \mathrm{EV}(M_1 \odot_v M_2)$. Then $x' \notin \mathrm{dom}(\Gamma) \cup \mathrm{BV}(M_1) \cup \mathrm{IVC}(M_1) \cup \mathrm{EV}(M_1)$, $x' \notin \mathrm{dom}(\Gamma\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}) \cup \mathrm{BV}(M_2) \cup (M_2) \cup \mathrm{EV}(M_2)$ and $x' \notin \{x'_i \mid i = 1 \cdots n\}$. Therefore, by the induction hypothesis, we obtain $\Gamma\{x':\sigma\}, \{x'/x\}\Delta_1 \vdash \overline{\{x'/x\}}M_1 : [\{x'_1:\sigma_1, \ldots, x'_n:\sigma_n\} \triangleright \tau_1] \Rightarrow \tau_2$ and $\Gamma\{x':\sigma\}\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}\{x'/x\}\Delta_2 \vdash \overline{\{x'/x\}}M_2 : \tau_1$.

Then, we arrive at $\Gamma\{x':\sigma\}, \{x'/x\}\Delta_1; \mathrm{Cl}(\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \{x'/x\}\Delta_2) \vdash \overline{\{x'/x\}}M_1 \odot_v \overline{\{x'/x\}}M_2 : \tau_2$ by the rule(fill).

Since $x \not\equiv x_i$, $\overline{\{x'/x\}}M_1 \odot_v \overline{\{x'/x\}}M_2 = \overline{\{x'/x\}}(M_1 \odot_v M_2)$ and

$\{x'/x\}\Delta_1; \mathrm{Cl}(\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \{x'/x\}\Delta_2)$

$= \{x'/x\}(\Delta_1; \mathrm{Cl}(\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \Delta_2))$. □

Lemma 11. *If* $\Gamma_1; \Gamma_2\{x'_1:\sigma_1, \ldots, x'_n:\sigma_n\}, \Delta_0 \vdash M_0 : \sigma$,
$\Gamma_1\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \Delta\{X:([\Gamma_2 \triangleright \sigma], \{x_1/x'_1, \ldots, x_n/x'_n\})\} \vdash M : \tau$,
$(\mathrm{BV}(M) \cup \{x_i \mid i = 1 \cdots n\}) \cap (\mathrm{IVC}(M_0) \cup \mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$ *and* $\mathrm{BV}(M) \cap (\mathrm{dom}(\Gamma_2) \cup \{x'_i \mid i = 1 \cdots n\}) = \emptyset$, *then*
$\Gamma_1\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \Delta; \mathrm{Cl}(\Gamma_2, \{x_1/x'_1, \ldots, x_n/x'_n\}\Delta_0) \vdash M[M_0/X] : \tau$.

Proof. We proceed by induction on the derivation of

$\Gamma_1\{x_1:\sigma_1, \ldots, x_n:\sigma_n\}, \Delta\{X:([\Gamma_2 \triangleright \sigma], v)\} \vdash M : \tau$.

$$\textit{Case}\ \ (\text{abs})\ \ \frac{\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}\{x:\tau_1\},\Delta'\{X:([\Gamma_2\triangleright\sigma],v)\}\vdash M_1:\tau_2}{\Gamma_1\{x_i:\sigma_i\},\Delta\{X:([\Gamma_2\triangleright\sigma],v)\}\vdash\lambda x:\tau_1.M_1:\tau_1\to\tau_2}$$
$$(v=\{x_1/x_1',\ldots,x_n/x_n'\},\ \Delta=\mathrm{Cl}(\{x:\tau_1\},\Delta')\ \text{and}\ x\not\equiv x_i).$$

Suppose $(\mathrm{BV}(\lambda x:\tau_1.M_1)\cup\{x_i|i=1\cdots n\})\cap(\mathrm{IVC}(M_0)\cup\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0))=\emptyset$ and $\mathrm{BV}(\lambda x:\tau_1.M_1)\cap(\mathrm{dom}(\Gamma_2\cup\{x_i'|i=1\cdots n\})$. Then by Lemma 6, we have $\Gamma_1;\Gamma_2\{x_1':\sigma_1,\ldots,x_n':\sigma_n\}\{x:\tau_1\},\Delta_0\vdash M_0:\sigma$.
We obtain $\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}\{x:\tau_1\},\Delta';\mathrm{Cl}(\Gamma_2,v\Delta_0)\vdash M_1[M_0/X]:\tau$ by the induction hypothesis. Then by the rule (abs), we have
$\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta;\mathrm{Cl}(\Gamma_2\{x:\tau_1\},v\Delta_0)\vdash\lambda x:\tau_1.M_1[M_0/X]:\tau$.
Since $x\not\equiv x_i$ and $x\notin\mathrm{FVC}(M_0)\subseteq\mathrm{dom}(\Gamma_1;\Gamma_2\{x_1':\sigma_1,\ldots,x_n':\sigma_n\})$, and by Lemma 4, $\mathrm{Cl}(\Gamma_2\{x:\tau_1\},v\Delta_0)=\mathrm{Cl}(\Gamma_2,v\Delta_0)$.

$$\textit{Case}\ \ (\text{abs})\ \ \frac{\Gamma_1\{x_1:\sigma_1\ \ldots,x_n:\sigma_n\}\{x:\tau_1\},\Delta'\{X:([\Gamma_2'\triangleright\sigma],v\{x/x'\})\}\vdash M_1:\tau_2}{\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta\{X:([\Gamma_2\triangleright\sigma],v)\}\vdash\lambda x:\tau_1.M_1:\tau_1\to\tau_2}$$
$$(v=\{x_1/x_1',\ldots,x_n/x_n'\},\Delta=\mathrm{Cl}(\{x:\tau_1\},\Delta')\ \text{and}\ \Gamma_2=\Gamma_2'\{x':\tau_1\}).$$

Since $\Gamma_2=\Gamma_2'\{x':\tau_1\}$, we have $\Gamma_1;\Gamma_2'\{x':\tau_1\}\{x_1':\sigma_1\ \ldots,x_n':\sigma_n\},\Delta_0\vdash M_0:\sigma$. By the induction hypothesis,
we obtain $\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}\{x:\tau_1\},\Delta';\mathrm{Cl}(\Gamma_2',v\{x/x'\}\Delta_0)\vdash M_1[M_0/X]:\tau_2$.
Since $x\notin\mathrm{FVC}(M_0)\subseteq\mathrm{dom}(\Gamma_1;\Gamma_2\{x_1':\sigma_1\ \ldots,x_n':\sigma_n\})$ and by Lemma 4, $\mathrm{Cl}(\{x:\tau_1\},\Delta';\mathrm{Cl}(\Gamma_2',v\{x/x'\}\Delta_0))=\Delta;\mathrm{Cl}(\Gamma_2,v\Delta_0)$. By the rule(abs), we have $\Gamma_1\{x_1:\sigma_1\ldots,x_n:\sigma_1\},\Delta;\mathrm{Cl}(\Gamma_2,v\Delta_0)\vdash\lambda x:\tau_1.M_1[M_0/X]:\tau$.

$$\textit{Case}\ \ (\text{app})\ \ \frac{\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_1\vdash M_1:\tau_1\to\tau_2\quad\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_2\vdash M_2:\tau_1}{\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta\{X:([\Gamma_2\triangleright\sigma],v)\}\vdash M_1\ M_2:\tau_2}$$
$$(v=\{x_1/x_1',\ldots,x_n/x_n'\}\ \text{and}\ \Delta\{X:([\Gamma_2\triangleright\sigma],v)\}=\Delta_1;\Delta_2).$$

Suppose $\Delta_1=\Delta_1'\{X:([\Gamma_2\triangleright\sigma],v)\}$. By the induction hypothesis applied to M_1, the judgment $\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\Delta_1';\mathrm{Cl}(\Gamma_2,v\Delta_0)\vdash M_1[M_0/X]:\tau$ is derivable. The rest of this subcase is by the rule (app). The subcase of $\Delta_2=\Delta_2'\{X:([\Gamma_2\triangleright\sigma],v)\}$ is similar to the case above.

$$\textit{Case}\ (\text{hole})\ \ \Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},\{X:([\emptyset\triangleright\tau],v)\}\vdash X^v:\tau$$
$$(v=\{x_1/x_1',\ldots,x_n/x_n'\}).$$

Since $x_i\notin\mathrm{IVC}(M_0)\cup\mathrm{BV}(M_0)\cup\mathrm{EV}(M_0)$,
we have $\Gamma_1\{x_1:\sigma_1,\ldots,x_n:\sigma_n\},v\Delta_0\vdash\bar{v}M_0:\tau$ by the repeated applications of Lemma 10.

$$\textit{Case}\ \ (\text{habs})\ \ \frac{\Gamma_1;\Gamma_1',\Delta\{X:([\Gamma_2\triangleright\sigma],v)\}\{Y:([\Gamma_2'\triangleright\tau_1],\emptyset)\}\vdash M:\tau_2}{\Gamma_1;\Gamma_1',\Delta\{X:([\Gamma_2\triangleright\sigma],v)\}\vdash\delta Y.M:[\Gamma_2'\triangleright\tau_1]\Rightarrow\tau_2}$$
$$(\Gamma_1'=\{x_1:\sigma_1,\ldots,x_n:\sigma_n\}\ \text{and}\ v=\{x_1/x_1',\ldots,x_n/x_n'\}).$$

By the induction hypothesis, we obtain
$\Gamma_1; \Gamma_1', \Delta\{Y : ([\Gamma_2' \triangleright \tau_1], \emptyset)\}; \mathrm{Cl}(\Gamma_2, \nu\Delta_0) \vdash M[M_0/X] : \tau_2$. The rest is by the rule (habs).

$$\textit{Case} \quad (\text{fill}) \quad \frac{\Gamma_1; \Gamma_x, \Delta_1 \vdash M_1 : [\Gamma_y \triangleright \tau_1] \Rightarrow \tau_2 \;\; \Gamma_1; \Gamma_x; \Gamma_y, \Delta_2 \vdash M_2 : \tau_1}{\Gamma_1; \Gamma_x, \Delta\{X : ([\Gamma_2 \triangleright \sigma], \nu)\} \vdash M_1 \odot_{\nu'} M_2 : \tau_2}$$

$(\Gamma_x = \{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}, \Gamma_y = \{y_1 : \sigma_1', \ldots, y_m : \sigma_m'\},$
$\nu = \{x_1/x_1', \ldots, x_n/x_n'\}, \nu' = \{y_1'/y_1, \ldots, y_m'/y_m\}$
and $\Delta\{X : ([\Gamma_2 \triangleright \sigma], \nu)\} = \Delta_1; \mathrm{Cl}(\Gamma_y, \Delta_2))$.

If $\Delta_1 = \Delta_1'\{X : ([\Gamma_2 \triangleright \sigma], \nu)\}$, then by the induction hypothesis applied to M_1, $\Gamma_1; \Gamma_x, \Delta_1'; \mathrm{Cl}(\Gamma_2, \nu\Delta_0) \vdash M_1[M_0/X] : [\{y_1' : \sigma_1', \ldots, y_m' : \sigma_m'\} \triangleright \tau_1] \Rightarrow \tau_2$.
The rest is by the rule (fill).
Suppose $\{y_{a_i}' \mid i = 1 \cdots p\} = \mathrm{dom}(\Gamma_2) \cap \{y_j' \mid j = 1 \cdots m\}$, $\{y_{b_i}' \mid i = 1 \cdots q\} = \{y_j' \mid j = 1 \cdots m\} \setminus \{y_{a_i}' \mid i = 1 \cdots p\}$, p+q = m, $\Gamma_2 = \Gamma_2'\{y_{a_1}' : \sigma_{a_1}', \ldots, y_{a_p}' : \sigma_{a_p}'\}$ and $\Delta_2 = \Delta_2'\{X : ([\Gamma_2' \triangleright \sigma], \nu\{y_{a_1}/y_{a_1}', \ldots, y_{a_p}/y_{a_p}'\})\}$. Then, we have $\Gamma_1 : \Gamma_2'\{y_{a_1}; : \sigma a_1', \ldots, y_{a_p}' : \sigma_{a_p}'\}\{x_1' : \sigma_1, \ldots, x_n' : \sigma_n\}\Delta_0 \vdash M_0 : \sigma$. Since $y_{b_i} \notin \mathrm{dom}(\Gamma_1) \cup \mathrm{dom}(\Gamma_2) \cup \{x_k' \mid k = 1 \cdots n\} \cup \mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)$, by Lemma 6, we have $\Gamma_1\{y_{b_1} : \sigma_{b_1}', \ldots, y_{b_q} : \sigma_{b_q}'\}; \Gamma_2'\{y_{a_1}' : \sigma_{a_1}', \ldots, y_{a_p}' : \sigma_{a_p}'\}\{x_1' : \sigma_1, \ldots, x_n' : \sigma_n\}\Delta_0 \vdash M_0 : \sigma$. By $(\mathrm{BV}(M_1 \odot_{nu'} M_2) \cup \{x_k \mid k = 1 \cdots n\}) \cup (\mathrm{IVC}(M_0) \cup \mathrm{BV}(M_0) \cup \mathrm{EV}(M_0), = \emptyset$, $(\mathrm{BV}(M_2) \cup \{x_k \mid k = 1 \cdots n\} \cup \{y_{a_i} \mid i = 1 \cdots p\}) \cup (\mathrm{IVC})M_0) \cup \mathrm{BV}(M_0) \cup \mathrm{EV}(M_0)) = \emptyset$. Therefore, by the induction hypothesis, we obtain
$\Gamma_1; \Gamma_x; \Gamma_y, \Delta_2'; \mathrm{Cl}(\Gamma_2', (\nu\{y_{a_1}/y_{a_1}', \ldots, y_{a_p}/y_{a_p}'\})\Delta_0) \vdash M_2[M_0/X] : \tau_1$.
Finally, we obtain $\Gamma_1; \Gamma_x, \Delta; \mathrm{Cl}(\Gamma_2, \nu\Delta_0) \vdash M_1 \odot_{\nu'} M_2[M_0/X] : \tau_2$ by the rule (fill) in a similar way to the case of (abs). □

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, Explicit substitutions, J. Funct. Program 1 (4) (1991) 375–416.

[2] L. Dami, A lambda-calculus for dynamic binding, Theoret. Comput. Sci. 192 (2) (1998) 201–231.

[3] G. Dowek, T. Hardin, C. Kirchner, Higher-order unification via explicit substitutions, in: D. Kozen (Ed.), Proc. Logic in Computer Science, June 1995, pp. 366–374.

[4] J. Gosling, The feel of Java, Computer 30 (6) (1997) 53–57.

[5] M. Hashimoto, First-class contexts in ML, in: Proc. 4th Asian Comput. Sci. Conf. Lecture Notes in Computer Science, vol. 1538, Springer, Berlin, 1998, pp. 206–223.

[6] H. Herbelin, A λ-calculus structure isomorphic to sequent calculus structure, Proc. Annu. Conf. of the European Association for Computer Science Logic, CSL'94, 1994.

[7] W. Howard, The formulae-as-types notion of construction, in: J.P. Seldin, J.R. Hindley, H.B. Curry (Eds.), Essays on Combinatory Logic, Lambda-Calculus and Formalism, Academic Press, New York, 1980, pp. 476–490.

[8] A.K. Joshi, S. Kulick, Partial proof trees as building blocks for a categorized grammars, Linguist. Philos. 20 (6) (1997) 637–667.

[9] S. Kahrs, Context rewriting, in: Conditional Term Rewriting Systems, 3rd Internat. Workshop, CTRS-92 Proc. 1993, pp. 21–35.

[10] S. Lee, D. Friedman, Enriching the Lambda calculus with contexts: towards a theory of incremental program construction, Proc. Internat. Conf. on Functional Programming, ACM SIGPLAN Notices, 1996, pp. 239–250.
[11] R. Milner, Fully abstract models of typed λ-calculi, Theoret. Comput. Sci. 4 (1997) 1–22.
[12] R. Milner, M. Tofte, R. Harper, The Definition of Standard ML, The MIT Press, Cambridge, MA, 1990.
[13] G.D. Plotkin, LCF considered as a programming language, Theoret. Comput. Sci. 5 (1977) 223–255.
[14] M. Sato, T. Sakurai, R. Burstall, Explicit environments, in: Proc. Typed Lambda Calculi and Applications (TLCA'99), Lecture Notes in Computer Science, vol. 1581, Springer, Berlin, 1999, pp. 340–354.
[15] M. Takahashi, Parallel reductions in λ-calculus, Inform. and Comput. 118 (1995) 120–127.
[16] C.L. Talcott, A theory of binding structures and applications to rewriting, Theoret. Comput. Sci. 112 (1993) 99–143.
[17] J. Wells, R. Vestergaard, Confluent equational reasoning for linking with first-class primitive modules, August 1999, available from `http://www.cee.hw.ac.uk/~jbw/papers/`.
[18] N. Wirth, Programming in Modula-2, Texts and Monographs in Computer Science, Springer, Berlin, 1983.